EXPERIMENT - 12

Advanced Transaction Simulation: Deadlocks, MVCC, and Concurrency

This document demonstrates advanced transaction scenarios in MySQL using InnoDB: Deadlock simulation, MVCC-based concurrency, and comparison of locking vs MVCC. Use two concurrent sessions (Session A and Session B) in Nimbus or MySQL CLI to reproduce behavior.

Setup (Run Once)

```
DROP TABLE IF EXISTS StudentEnrollments;

CREATE TABLE StudentEnrollments (
  student_id INT NOT NULL,
  student_name VARCHAR(100) NOT NULL,
  course_id VARCHAR(10) NOT NULL,
  enrollment_date DATE NOT NULL,
  PRIMARY KEY (student_id)
) ENGINE=InnoDB;

INSERT INTO StudentEnrollments (student_id, student_name, course_id, enrollment_date)
VALUES
(1, 'Ashish', 'CSE101', '2024-06-01'),
(2, 'Smaran', 'CSE102', '2024-06-01'),
(3, 'Vaibhav','CSE103', '2024-06-01');
```

Part A: Simulating a Deadlock Between Two Transactions

Two concurrent transactions attempt to update overlapping rows in reverse order. MySQL detects a deadlock and rolls back one transaction (Error 1213).

Session A

```
START TRANSACTION;
UPDATE StudentEnrollments
SET enrollment_date = '2024-07-01'
WHERE student_id = 1;
SELECT SLEEP(5);
UPDATE StudentEnrollments
```

```
SET enrollment_date = '2024-07-02'
WHERE student_id = 2;
COMMIT;
```

Session B

```
START TRANSACTION;
UPDATE StudentEnrollments
SET enrollment_date = '2024-08-01'
WHERE student_id = 2;
SELECT SLEEP(5);
UPDATE StudentEnrollments
SET enrollment_date = '2024-08-02'
WHERE student_id = 1;
COMMIT;
```

Expected Output: One transaction fails with ERROR 1213 (Deadlock found...). The database automatically rolls back the victim transaction.

 To avoid deadlocks: always lock rows in the same order, keep transactions short, and retry on deadlock errors.

Part B: Applying MVCC to Prevent Conflicts During Concurrent Reads/Writes

MVCC allows non-blocking reads under Repeatable Read isolation. Session A reads a snapshot while Session B updates concurrently.

Session A (Reader)

```
SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
START TRANSACTION;
SELECT enrollment_date FROM StudentEnrollments WHERE student_id = 1;
-- Expect: 2024-06-01
SELECT SLEEP(5);
SELECT enrollment_date FROM StudentEnrollments WHERE student_id = 1;
COMMIT;
```

Session B (Writer)

```
START TRANSACTION;
UPDATE StudentEnrollments
SET enrollment_date = '2024-07-10'
WHERE student_id = 1;
```

COMMIT;

Expected Output: Session A sees old snapshot (2024-06-01) until commit, while Session B successfully updates to 2024-07-10.

Part C: Comparing Behavior With and Without MVCC

This demonstrates blocking behavior with explicit locking (SELECT FOR UPDATE) vs non-blocking reads with MVCC.

Scenario 1: With Locking

Session A:
START TRANSACTION;
SELECT * FROM StudentEnrollments WHERE student_id = 1 FOR UPDATE;
SELECT SLEEP(6);
UPDATE StudentEnrollments SET enrollment_date = '2024-09-10' WHERE student_id = 1;
COMMIT;

Session B:
START TRANSACTION;
UPDATE StudentEnrollments SET enrollment_date = '2024-09-20' WHERE student_id = 1;
COMMIT;
-- Session B blocks until A commits

Scenario 2: With MVCC (Non-blocking Reads)

Session A:
SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
START TRANSACTION;
SELECT enrollment_date FROM StudentEnrollments WHERE student_id = 1;
SELECT SLEEP(6);
SELECT enrollment_date FROM StudentEnrollments WHERE student_id = 1;
COMMIT;

Session B:
START TRANSACTION;
UPDATE StudentEnrollments SET enrollment_date = '2024-10-01' WHERE student_id = 1;
COMMIT;

Expected Output:
- With Locking: readers/writers block each other.
- With MVCC: readers continue using snapshot; writers commit new versions.

Sample Output Image

Part A: Insert Multiple Fee Payments in a Transaction

Set fee_amount date

START TRANSACTION tennne

payment_id	student_name	amount	amn	payment_date
1	Ashish	5000.00	050	2024-06-01
2	Smaran	4500.00	030	2024-06-02
3	Valbhav	5500.00	083	2024-06-03

COMMIT

Part B: Demonstrate ROLLBACK for Failed Payment Insertion

START TRANSACTION

payment_id	student_name	amount	amn	payment_date
4	Kiran	4000.00	400	2024-06-04
5	Smdran	-100.00	-100	2024-06-05

ROLLBACK

Part C: Simulate Partial Failure and Ensure Consistent State

START TRANSACTION

payment_id	student_name	amount	amn	payment_date
5	Nidhi	3000.00	000	2024-06-08
6	Smaran	2500.00	000	2024-06-07
3	Valbhav	5500.00	000	2024-06-07

ROLLBACK

Part D: Verify ACID Compliance with Transaction Flow

START TRANSACTION

payment_id	Ashish	5000	5000.00	Fe	233-0865	2024-06-01
payment_id	Smaran	4600	4500.00	M0	234-0035	2024-06-02
payment_id	Vaibhav	5500	5500.00	22	654-0085	2024-06-03

RE0nT TRANSACTION SELE Inalict transactions for 2024-06-05